An introduction to OpenStack

This is the write up of a talk I gave to Linux Users Victoria, Australia, on December 4th, 2012, introducing the members to OpenStack. It reflects my understanding of how OpenStack works – and there is still a lot that I don't understand or know. So caveat emptor – and all corrections gratefully accepted!

My Success metric

If I can get attendees to understand that OpenStack is rapidly growing, modular, to understand the role of each component, and then to go and download it and play with it I'll judge my efforts to be a success.

Games

The games were activities to be thrown in, if time permitted, to break any possible tedium...

Document version: 1.0

Introduction

Hi all. My name is Martin and I am a Java developer.

Who has worked on both sides of the OpenStack cloud.

I have worked on the NeCTAR OpenStack deployment, helping by hunting down bugs, and adding some needed customisations.

And I have worked with the VLSCI on attempting to get Galaxy, a large genomics application written in Python up and running on the NeCTAR cloud.

On both projects I found myself in the world of Linux, Python and virtual machines. Given my Java on Windows background I found myself often feeling like a hamster on a wheel as I rushed to educate myself.

When I wasn't educating myself I had my nose buried deep in the guts of the machine, looking at fine details. So my world view was either way out there, or looking at the microscopic detail.

Given my background, and given that this is the Linux User Group, I expect you all to be far more knowledgeable than I in the tools and technologies that make up OpenStack! But the good news is that if I, given my background, was able to cope, you all should be able to master OpenStack in short order!

So how did a Java developer land up working in the OpenStack world? Well, shortly after I joined my current employer, VeRSI, they ask for a volunteer. And believing implicitly in the cloud, I jumped up and down, shouting “Pick me!” “Pick me!”. And so I found myself put to work on OpenStack.

A Story

But why my enthusiasm about the cloud?

Well, I used to be such a doubter. I thought it was all just was just marketing hype.

Then about three years ago the company I then worked for placed me as a consultant in the roll of technical team lead on a project developing a new website for a University.

The team and I had a hard deadline of two months to deliver the new site, on a new CMS, with new content and a new look, on new hardware.

The first thing I did was calculate the number of servers that the team and I would need for the project.

I picked up the phone to the university's IT services department.

“Could I have 7 servers to go please?”

The voice on the other side cackled: “You're new here, aren't you? Go to our website, download the application form, fill it in in triplicate, mail the copies to the acquisitions committee, who will then either approve or decline your request. If approved, the request goes out to tender, the successful bidder is selected and an order placed. Eventually you will get your servers.”

“How long does this take” I asked, feeling faint.

“The committee sits once a month, and they've just sat, so you'll have to wait a month, and then ordinarily it takes at least 4 weeks for the hardware if the purchase is approved. So for you, at least 8 weeks”

I hung up and buried my head in my hands. We were totally screwed. There was no way that we could make our deadline!

Then the Universities development lead said “I know a site that will host our source code and defect tracking system for a few dollars a month. Someone lend me a credit card”.

Inspired by his example I thought “Amazon!” I phoned my companies hardware and network guru to talk it through.

“Don't worry about Amazon - I've got a big server lying spare” he said. “I'll set up some virtual servers for your team on it. How many do you want?”

30 minutes and $10 later we had all the hardware and software we needed and by the end of the day we were already committing code and accepting defect reports.

In a blink of an eye the politics, processes and policies of the University had been totally upended.

When physical servers eventually arrived their cost was in the order of about $20 000. And they arrived well before the end of the project, because we threatened to go live on our virtual servers, thus demonstrating that the IT services department wasn't really needed...

That project showed me that despite my doubts, cloud computing is a game changer. Everybody in IT, no matter what their role needs to understand this.

Buzzword Bingo

Cloud computing comes in so many different flavours and models that it can be quite confusing.

To me cloud computing is simply the use of computing resources (both hardware and software) that are delivered as a service over the internet.

And there are three important ways of delivery that we have to understand:

- SaaS: Software As A Service Synonym: Application – End Users
- PaaS: Platform As A Service Synonym: API – Developers
- IaaS: Infrastructure As A Service Synonym: Hardware – System Administrators

Software as a service means that you are interacting with a cloud based application, ordinarily by means of a web browser. However, the trend is, especially in the mobile space, toward native clients. Examples of SaaS? GMail, Dropbox and SalesForce.

Platform as a service means that you are creating an application around an API or library that abstracts the underlying cloud infrastructure away. Examples of PaaS? Google's AppEngine, AppScale, Heroku and Engine Yard.

Infrastructure as a service: providers offer computers and storage as resources. You provision these machines with operating systems and software yourself. The most famous of these providers is of course, Amazon.

IaaS is the cloud computing plane that OpenStack lies in.

Game: *aaS

We have too many “aaS”'s in this industry. Any guesses as to the following:

BaaS: Backend as a Service	You write the client, we provide the server side stuff. Could also be “Backup as a Service”
DRaaS: Disaster Recovery as a Service	
DBaaS:	Database as a service.
CaaS: Compute as a Service	What's the difference between this and IaaS?
EaaS: Everything as a Service	Yes! Really. WTF?
STaaS: STorage as a Service	“You bring it, we store it”[1]
XaaS:	“X” as a Service. Yes, an algorithm for tired IT professionals.
ITaaS:	IT as a service.
ZaaS:	Zombies as a Service[2]

1 “As a proud Family Business you can feel comfortable in storing with People Who Care. We have storage locations everywhere! And space for everyone! Personal and Business Space, Wine Cellars, RV and Boat, Gun Storage and Deposit Boxes. Keep it at Kennards, the People Who Care!”

2 I made this last one up as joke to finish this game, but then was told some security researchers use this term to describe compromised computers available for hire...

Virtualization

A common model for IaaS providers is to make use of hardware virtualization, in which a single physical server hosts many virtual machines, each of which is running their own private operating system.

This is the model followed by Amazon, and its the model followed by OpenStack.

Virtualization is a thing of wonder to me.

For example, you can install Linux into VirtualBox, on your desktop, and then install OpenStack into that Linux instance, and then within that OpenStack installation, launch a VM and then install another OS into that instance...

The whole thing reeks of Russian Dolls, and boggles my brain.

If you think about it, having all those virtual machines sharing one physical cpu means that your application running on one of those virtual machines may well be impacted by what's happening in the other virtual machines – especially with regard to input and output[3].

I love the term used to describe this affect: “Noisy neighbours”.

Another feature of virtualization that boggles my brain is the ability to migrate a running virtual machine from one physical machine to another. The wonder! This live migration allows the IaaS provider to optimise the utilisation of their physical servers.

Another set of concepts to be aware of are “Ephemeral” volumes and “Persistent” volumes.

An Ephemeral volume is storage that simply vanishes when it's associated virtual machine is terminated. Ordinarily, on OpenStack, the virtual machines root file system is stored on the ephemeral storage[4].

Rebooting or restarting its associated virtual machine does not cause the ephemeral storage to vanish.

But remember, what ever you write to this ephemeral storage will be lost the minute your virtual machine is terminated. All upgrades, security patches, whatever you write, all will be lost.

I've seen ephemeral storage take so many people by surprise when they first start using OpenStack![5]

Persistent volumes on the other hand are independent of any given virtual machine instance. When first created they have no partition, and no file system. They must be explicitly attached to an instance, and then be partitioned and formatted. Then they can be used as though they were an external hard drive.

Persistent volumes are where you would want to store data that has to survive between virtual machine runs.

Given the above you can see that moving existing applications to the cloud will require a certain amount of rework. You can't just take an existing web application and shove out to the cloud and expect it to be as performant and reliable as it was on dedicated hardware.

As a software developer, this is a personal pain point. If you hear people say “Oh, we'll just move it out onto the cloud” be prepared to start to argue with them. These are hype-tevated people, and they

3 A side effect is that you can't take a virtualized operating system's statistics at face value.
4 OpenStack provides an additional ephemeral raw block device that your virtual machine can mount format, and make use of. Some OS's, such as Ubuntu, will automatically format and mount this space (/mnt)
5 As a side effect you should plan to keep on grooming your virtual machine images!

need a cold dose of reality poured into their world.

Game: What would Google do?

Google's servers: somewhat different to yours and mine...

Do they use virtualization?

Do you think you can compete with them?

What would it take to compete?

The history of OpenStack

So what is OpenStack?

At it's heart it's really just a lot of Python scripts that glue together a raft of other, more mature, projects to create an open source IaaS cloud computing implementation that makes heavy use of virtualization.

How did it come into being?

Over a candlelit dinner, just over two years ago, Nasa and RackSpace looked deeply into each others eyes, and decided to combine two different system, a storage component from RackSpace, and a virtual machine manager from Nasa named Nebula, and to release them to the world under an Apache 2 licence.

To me RackSpace's storage component was mature and in production use, Nasa's cloud component was young, and not in such widespread use. Luckily for this match, they were both written in Python.

As a point of interest, one of the reasons that Nasa wrote Nebula was that they felt that Amazon was targeted at web hosting, and they needed to work with very large “science scale” data sets. So Nebula scales – it was used with Nasa's large data sets.

Once the project was created the community grew as other companies joined it. One of the first outside companies to become involved was Canonical, the makers of Ubuntu.

When Canonical signed up it appears that they threw both developers, infrastructure and resources into the project. As a result, Ubuntu is the OpenStack operating system of choice, Canonical's Launchpad provides the project management, and the two projects release cycles are synchronised.

So OpenStack thus has a six monthly release cycle, each release cycle being kicked of with a design month that includes a summit meeting held somewhere suitably exotic. The design month is followed by an implementation phase, then everything is locked down for the usual release candidate dance.

The names of the releases are somewhat bewildering:

Austin, Bexar, Cactus, Diablo, Essex, Folsom and Grizzly, currently under development.

That's because these names are selected by choosing “cool sounding”[6] city or county names that are near to the site of the design summit meeting as candidate names, and then having a poll to select the winning name.

Initially, OpenStack planned to support Amazon's API's. But as time progressed I believe that it came to view these API's as a strategic weapon armed in Amazon's favour – and so does not spend a great deal of time and effort in trying to make them 100% compatible with Amazons.

Since those early days other companies have become involved. Big ones, like AT&T, IBM, Hewlett Packard, Dell, Cisco.... Why, even Microsoft has contributed code.

In its short lifetime, OpenStack has grown to be, IMHO, a rather large and exciting project. There are over 180 companies involved, and over 400 developers beavering away on the code in one form or other. In order to support this growth RackSpace ceded control to a foundation that now steers the project.

6 Their words, not mine.

So Be Prepared: History teaches us:

- OpenStack is young, some parts are reasonably new, and the internal interfaces and naming conventions between its components aren't that consistent yet.
- If you want to have the least pain when you get started, use Ubuntu.
- If you write to either Amazon or OpenStack's API's, you are potentially locking yourself in to a pan galactic strategic API battle. Use a library to abstract those pesky API's away, if you can.
- If you alter the OpenStack code without getting the community to accept your changes you're in for a world of pain.
- And if you put it into production use, be prepared to upgrade every six months or so!

OpenStack: the components

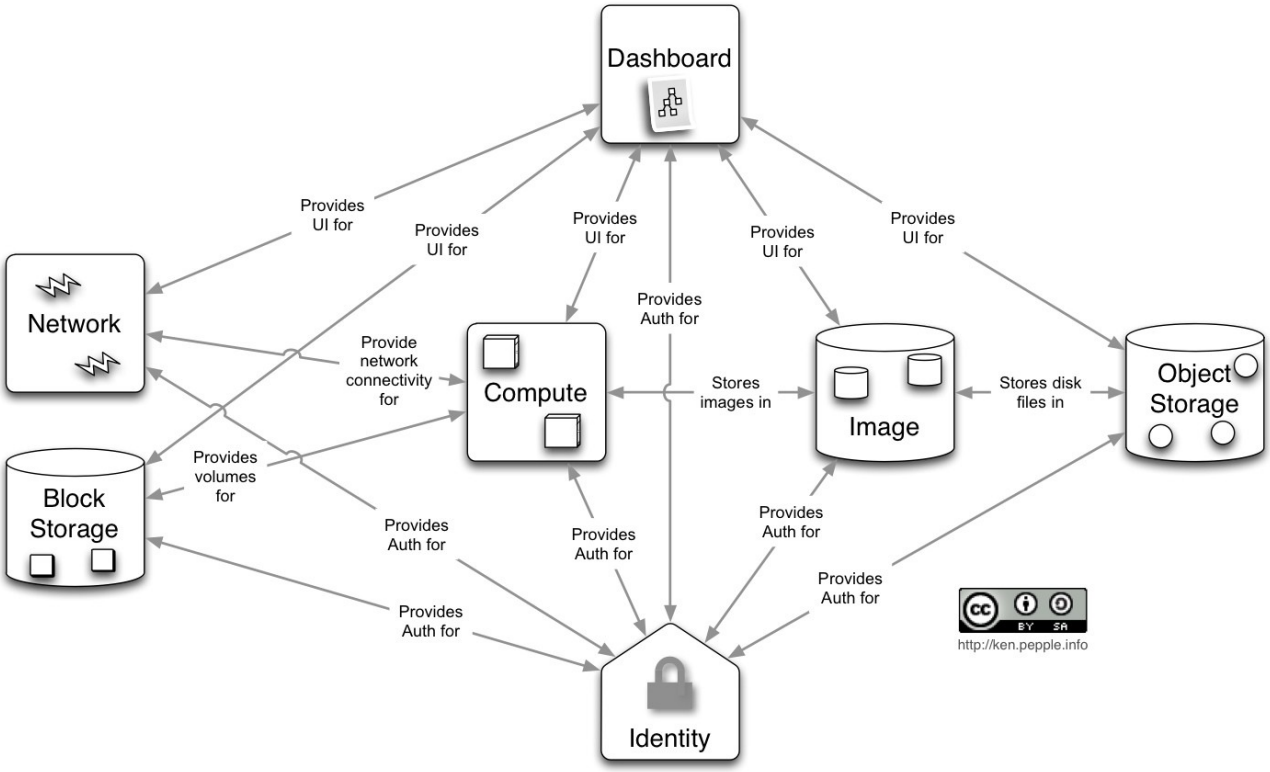

Illustration 1: OpenStack Components, diagram by Ken Pepple (http://ken.pepple.info/)

OpenStack is not a single piece of software: it is a collection of components, that when used together, will allow you to create your own cloud stack.

As such it is based on a set of separate development projects. The code developed by each project targets a specific role within OpenStack. In typical fashion, each project has their own project name.

Dashboard is the web front end that gives people a nice GUI to interact with the system. Dashboard's OpenStack project name is “Horizon”.

Compute manages the startup, provision and shutdown of the virtual machines with in the system. Compute's project name is “Nova”.

The Object Storage is a fault tolerant redundant file store. Its project name is “Swift”.

The Image Store is a catalogue/repository for disk images. Its project name is “Glance”.

Block Storage simply provides a block storage for the running virtual machines. In essence it's simply providing blank hard drives to the applications on the virtual machines to use as they will. Its code name is “Cinder”

Network provides a software defined networking platform – thus giving network as a service. Its project name is “Quantum”

Underpinning it all is the identity service, named “Keystone”

To me, based on the amount of swearing I heard at NeCTAR, the effort involved in getting an OpenStack deployment set up is spent in the configuration of all these components to work together.

BTW, because of this modular design you can use a number of these components in their own right.

To map this all to Amazon's services:

- Compute/Nova is similar to EC2
- Object Storage/Swift is similar to S3
- Image/Glance plays the roll of the AMI catalogue
- Block Storage/Cinder is similar to EBS

OpenStack: the architecture

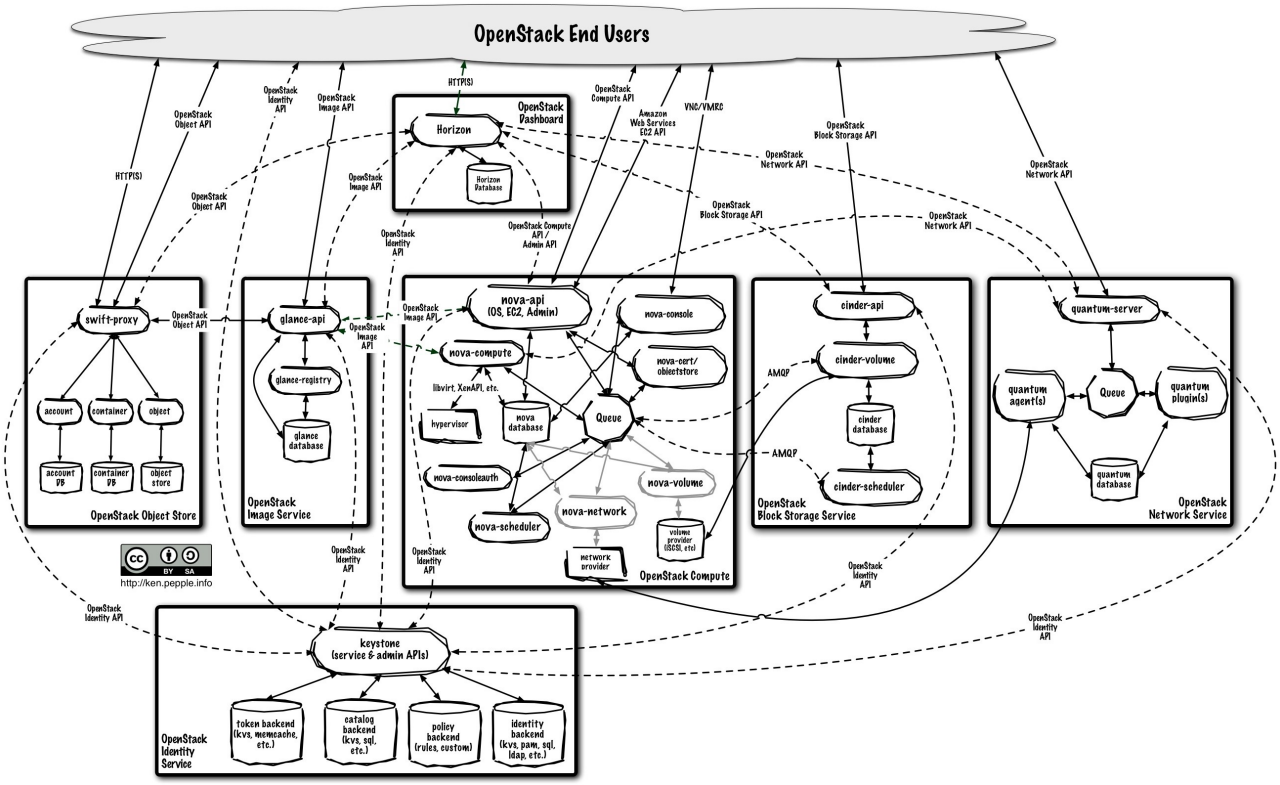

Illustration 2: An Example Logical Architecture, diagram by Ken Pepple (http://ken.pepple.info/)

[Print this diagram out and refer to it in the sections that follow]

Given the modular nature of OpenStack, and the fact that not all components need to be deployed, the logical architecture of a typical OpenStack deployment is a far more complex beast.

Users can interact with OpenStack either through the Dashboard web interface, or directly with each component through that components API (straight lines in the diagram). There is a matching command line client for each of the components.

Individual components will mostly interact with each other through their public API's.

Internally, a number of the components make use of message passing though a queue in order to execute the requests received via the public API.

In these components if the API receives a request, it may pop a message on a queue and respond with an “OK”.

If this was done at some later time a worker thread will pick up the message, and act on it. The worker may in turn put more messages on the queue for other worker threads to respond to.

All the components authenticate by means of Keystone.

The API's generally have an admin API and a user API that can be configured to run on different ports. Its considered good practice to make sure that the admin API is not visible to the outside world!

It's important to know that the API's are basic restful http web services. These web services are an implementation of the Python WSGI specification, each web service written as a combination of middleware and applications working together to create the API.

The Python WebOb library is used within the API implementations to provide a layer of abstraction over the raw WSGI and HTTP specifications.

The Python Routes library is used to map the URL's that make up the API to the applications that will handle the calls.

Finally, Python Paste Deployment serves to configure the code that makes up the API.

This API implementation thus allows extension if you have the courage to climb in and customise it. And the implementations have been written to support such customisation.

The worker threads are based on the Python Eventlet library: An event driven green threaded workers union.

So as you can see OpenStack is really a highly modular, highly configurable and extensible set of Python scripts!

If you are familiar with “The Innovator's Dilemma” by Clayton Christensen, OpenStack appears to have what it takes to be very disruptive.

Keystone (Identity Service)

Keystone provides an Authentication and Authorisation API.

In this role, Keystone's design goal was simply to be a shim in front of existing external user systems, such as LDAP Out of the box, the external user systems supported are:

- key/value stores,
- SQL
- PAM
- LDAP
- templated

The expectation is that these aren't suitable, you will write your own plug in to fit your infrastructure.

Importantly, Keystone also serves as a catalogue of the services within its associated OpenStack installation.

So when KeyStone first starts up you essentially have to describe the entire OpenStack installation to it in terms of the API url's (endpoints, in Keystone talk).

Why combine these two services?

Well, lets go through a typical usage scenario.

You log on, passing Keystone your user id and password. To do this the software you are using needs to know the Keystone URL, obviously. But that is the only URL it ever needs to know in order to make use of the OpenStack installation.

Because what happens next is that Keystone validates your credentials against the external system, and if successful returns what is called an “un-scoped token”.

The software you are using then uses the un-scoped token to request from Keystone the details of the services that you are allowed to access.

Having got that list, the software then selects the service that you want to call and then sends another request to Keystone, again with your un-scoped token, asking for that particular services's details.

Keystone replies, giving that services's base URL and also what is called a “scoped token”. The base URL is combined with the URI's of the API to create the URL of the API call you want to make.

The call is then made using the associated scoped token.

When the service receives your call, it takes the scoped token and asks Keystone if the token is valid or not. If Keystone approves, your request is accepted. If not, it is rejected.

The tokens have an an associated life time, so if you try to use one beyond that time period, you are forced to re-authenticate.

So really Keystone can be seen as a generic cookie monster that manages the following services:

- Identity service: credential validation, and information on Users, Tenants (Projects) and Roles.
- Token service: validates and manages tokens.
- Catalogue service: endpoint registry and endpoint discovery

- Policy service: rule management and authorisation engine

As you can see this is all incredibly chatty, and thus somewhat limiting. Hence a more efficient public key infrastructure solution has been been introduced in the latest release.

Under the PKI solution, Keystone encrypts its responses and signs the tokens using a private key. When the other components are first called, they request Keystones public key, which they then cache. From that point on they can simply check the signature on the token to confirm its validity. A far more performant and secure solution!

Swift (Object Storage)

Quote:
Brandon Hays (@tehviking): “Okay, geeks, we got everyone saying 'the cloud'. Phase 2: get people calling databases 'the cylinder'”

The Object Storage component, named “Swift” is a distributed, highly fault tolerant, redundant, eventually consistent file store.[7]

It is the RackSpace contributed portion of the original OpenStack project, all those two and a bit years ago. It's the piece that is most battle tested. It's thus also the piece that's likely to have slightly different API patterns, calls and terminology. But it is being brought into line.

It's important to remember that Swift is not a file system! Its an object store designed to work with clusters of commodity servers, storing static data, such as photo's, images, and documents, on a long term basis.

Swift knows only about accounts, containers and objects.

Each user gets their own account, and has full access to that account.

Within their account, users create containers.

Within a users account there can't be duplicate container names. However, other accounts can reuse container names. It's not like Amazon where all bucket names have to be unique across the known universe.

Containers hold objects. Not containers, only objects. So there are no nested container hierarchies. You create a container, and you put one or more objects into it.

Container names can't contain '/' and are limited to 256 bytes in length after URL encoding.

The object that you upload into your container is stored 'as is'. Its not encrypted or compressed.

Every time you want to update an object, you have to overwrite it. Thus objects in Swift are effectively immutable.

The only restriction on object names is that they are limited to 1023 byes in length after URL encoding.

Uploaded objects must be less than 5 Gig in size.

Bizarrely, you can upload an object of 0 bytes in size!

Access to the uploaded objects and containers can be widened by account and user, and also to the whole wide world.

Remember, in keeping with the other OpenStack API's, Swift is accessed by a restful http API. Thus all objects in Swift can be accessed by means of a URL. So you can use Swift to serve up images and other static content for a web site.

You can tag an uploaded object with metadata, in the form of up to 90 key value pairs, with a combined length of not more than 4k. Don't depend on the case of the text in your key value pairs.

There is no query language, and now way to search on meta data. So if you want to index objects, you need to do the indexing externally.

Under the hood the servers that make a Swift installation are allocated to zones. The

7 In keeping with the CAP theorem.

recommendation is to set up at least five zones.

When an object is written to Swift, it has a unique hash associated with it and is then copied to at least three of the zones.

So beware: if you upload a new version of an object and then do an immediate GET, you might land up with an older copy! The copies will eventually become consistent. If this worries you, you will have to add and use version metadata when you upload objects.

Quote:
JRR Tolkien: “One Ring to rule them all, One Ring to find them, One Ring to bring them all and in the darkness bind them”

Overseeing the whole process is something termed the “ring”. Technically a consistent hash ring, for our purposes, a fixed map to show where objects should be stored and retrieved within a Swift installation.

Auditor processes continually scan the stored objects, and if corruption is detected, the object is moved to a quarantine area.

Replicator processes also continually run, and if they find missing files or files that don't have the mandatory number of copies they will make the required new copies.

If a zone dies, replicators in the other zones containing copies will notice and proactively make copies of the “lost” objects.

Internally, the network in a Swift installation is thus incredibly chatty.

Game: Could you, would you...

Could you, would you, set Swift up to run on your collection of Raspberry Pi's? After all, what could be cooler than a fault tolerant, redundant file store in a large lunch box?

Cinder (Block storage)

The Block Storage component, named “Cinder”, simply provides a persistent block storage service for the running virtual machines.

The API receives requests, then passes them on by means of a message queue, to worker process that will act on the requests.

This component uses a database to maintain state in.

At its heart it is simply an iSCSI solution that employs LVM

It has a pluggable architecture, that allows different storage devices to be supported.

For example:

- iSCSI
- IBM
- SolidFire
- NetApp
- SheepDog
- Nexenta
- Zadara
- Others...

When creating a volume, Cinder tries to pick the best underlying block storage node to create a volume on.

Snapshots of the block storage volumes are made using LVM snapshot. Hence not only are these incremental: you probably don't want to leave snapshots lying around. This is not official advice: just me, based on my understanding of LVM snapshots.

Support has been added for booting from a volume, but I'm not too certain about its state.

Here are some general tips from personal experience when working with the block storage.

- It's essentially a mountable volume, so its not sharable between virtual machines. If you want to share it between running virtual machines, you need to unmount it from the first machine, then mount it on the next.
- Remember, when unmounting block storage, if you have database on it, shut your database down first!
- Snapshots can be taken.
- But stop all database writes before making a snapshot!
- Once you've made your snapshot, work with it and then discard it (me again).

Glance (Image Store Component)

The Image Store component, named “Glance” is simply a repository for disk images.

It was contributed to OpenStack way back when, and is possibly the most stable and unchanging of the OpenStack components.

Via its API it provides discovery and registration services for virtual disk images. Via the client library you can store and retrieve the images.

It stores metadata about the images in a database, and uses a repository to store the actual images. The repository is pluggable, meaning that you can choose how and where the images are stored.

The choices are

- Swift (of course)
- A regular filesystem
- RADOS block devices
- Amazon's S3
- HTTP

Glance supports caching and replication, so that you can be assured that if you have a cluster the disk images will be consistent across it.

Glance supports different image formats. These are the formats currently supported:

- Raw (an exact bit copy of a hard disk)
- Machine (kernel/ramdisk outside of the image, ie: AMI)
- VHD (Hyper-V)
- VDI (VirtualBox)
- qcow2 (Qemu/KVM)
- VMDK (VMWare)
- OVF (VMWare, others)

The images that you upload can be marked as private, or can be shared with the world (public).

If you are using KVM or Xen as your hypervisor, you can take a snapshot of a running instance – which will actually create a full new private image in glance. If, however, you have a volume attached to your running instance when you try to do this, it all turns to custard...

You can use Glance to set up a private/trusted repository of images, thus giving:

- your developers and users a head start.
- backups
- preconfigured images to customers (WordPress to go, for example)

Quantum (Network)

I'm a software developer, so the complexities of networking are something that I've never really had to bother about. Given that we are moving to a world of software defined networking, with separate control and data planes, this is possibly going change. Yet another thing to learn on that great hamster wheel of self development!

Again, networking in OpenStack is a highly configurable component, and again, it has a plug-in architecture.

The intent is that vendors of networking equipment will write their own plugins that will interact with their equipment.

The plugins that I'm aware of at this moment in time are for:

- Cisco
- Nicira NVP
- Nec OpenFlow,
- Open vSwitch,
- Linux bridging,
- Ryu Network Operating System
- Midokua

I'm certain that this list is growing all the time, given the number of people who seem to be installing and playing with OpenStack.

This vendor provided layer is responsible for the plugging in and unplugging of ports, creating networks, or subnets and managing the IP addresses.

The API receives requests, and then passes them on to the appropriate plugins for action, once again, ordinarily by means of a message queue.

The database is used simply to store the network state in, mainly for use by the plugins.

So this is really a software defined networking application that is largely dedicated to managing the dynamically changing networking needs of the virtual machine instances in the compute (Nova) cloud.

Horizon (Dashboard)

The dashboard component, named 'Horizon', is simply a Django web application that provides a simple GUI for end user's and administrators.

It talks to the other OpenStack components via their API's, so although it has a database it stores very little data in it.

When I first saw the configuration file I was struck by how empty it was. It basically consisted of the database connection details and the url for the Keystone server.

Its very pretty, and is useful if you like GUI's.

Game:

Why only the Keystone server url in the configuration file?

Nova (Compute)

The Compute component, named Nova, manages the startup, provision and shutdown of the virtual machines with in the system.

Really, it's simply a hypervisor orchestrator that draws on the components we've covered so far.

So by now the OpenStack pattern should be very familiar:

A restful web API that will put commands onto a queue when called. A set of worker threads that will accept the commands and perform some action, then either terminate or place another command on the queue.

An underlying database is used to maintain both build and run time state.

And of course the underlying hypervisor is pluggable, so when you set up you OpenStack install can choose one of:

- Xen
- KVM
- QEMU
- LXC
- ESXi
- Hyper-V

KVM is the default hypervisor used in development so I'd used it when playing with OpenStack.

There is a schedular to determine where virtual machines should be run (pluggable, of course)

And there is a console service to allow end users to access their vm's console via a proxy.

BTW, Instances have their inward bound network packets filtered by IP Table rules. As a user you can create a named set of rules, termed a 'security group' to be applied to your running instance. If you don't select a security group when you fire your instance up, a default security group will be used.

All outbound network traffic is allowed. If you don't want this, you have to configure a firewall inside your instance.

Now, hopefully, we are going to tie the whole thing together.

You're user and you want to launch a virtual machine. What do you do?

We assume your logged and you have a scoped token.

You have to call the Nova API with a run instance command, usually through Horizon or the command line client.

Nova API takes the scoped token and confers with Keystone.

Permission being granted, the API then creates a security group

Generates the MAC address and hostname

Sends a message to the schedular to run the instance.

The schedular chooses the target host, and casts a message onto the queue (its not a call!)

In time the compute work thread receives the message and checks to see if the instance isn't already running, and aborts if it is.

Then allocates a fixed IP address via a call to Quantum. And a VLAN and a bridge.

A configuration file (libvirt.xml) describing the image is prepared for the virtualization driver.

The disk image is copied from Glance to a file on the host os.

The SSH keys are injected into the image by mounting the file and writing to it.

The file is unmounted, and handed over to the virtualization driver to be run.

By the way, when the command to shut down that instance is received, that file that was used as the image is simply thrown away. Hence the ephemeral storage.

Why do you want to play with OpenStack?

Why do you want to play with OpenStack?

Lets see if we can count the ways.

First off, there's those design meetings in exotic locations. If you can get your company to adopt OpenStack you'll have to have have some input at those design meetings.

Then, it's very easy to get going. OpenStack allows you to set up a cloud environment in a virtual machine, on your desktop. You can start exploring cloud computing now, with your credit card safe in your pocket.

And don't forget: some of the components can be spun off and used in their own right.

From a development perspective, you can put applications you are creating for the cloud into a controlled environment and destroy them – easily, bit by bit. You can then understand better how they'll behave, and how they will scale.

Because so many different vendors are adopting OpenStack, if you adopt it as well, it means that you will not be locked into a single cloud provider: you can avoid single sourcing your cloud provider!

This might also focus minds:

How to get going

So how do you get going?

If you want to install OpenStack, your first stop should be DevStack (http://devstack.org). It contains a set of scripts that will install OpenStack into a number of different environments. For your first attempt, spin up VirtualBox, install Ubuntu 12.04, and inside that running instance clone and run the DevStack script. What could be simpler?

If you simply want to how applications perform under OpenStack, or interact with the API's via the command line, your first stop should be TryStack (http://trystack.org). It's a set of OpenStack installations on both ARM and x86 that are there for this purpose. Accounts on TryStack are regularly wiped to make sure people aren't actually running production software on it!

The Australian OpenStack User group (http://aosug.openstack.org.au/) is a wonderfully enthusiastic local community. If you join up, you'll be meeting some of the local early adopters, and you'll be able to learn from them.

Then you can explore the OpenStack home site: http://www.openstack.org . There you'll find a cornucopia of links to more detailed information: wiki's, code, IRC, newsgroups, how to submit bugs, the project blog and the documentation.

BTW, OpenStack is one of the better documented OpenStack projects that I've encountered. One issue is that the rate of change is so fast, the documentation team sometimes struggle to keep up to date. Also make sure that the version of the documentation you are looking at matches the version of OpenStack you are playing with!

Remember, contributions are welcomed by the OpenStack community!

Conclusion

We consistently have favoured convenience and features over quality and time. Windows vs OS2, JHS vs Betamax. The history is clear. The cloud offers both convenience and features in spades. We will be moving to it. We must be prepared.

So give OpenStack a spin today!